PATENT ABSTRACTS OF JAPAN

(11)Publication number : 08-227405
(43)Date of publication of application : 03.09.1996

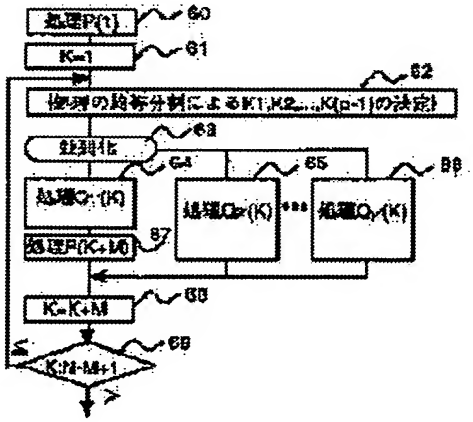
(51)Int.Cl. G06F 15/16
G06F 17/12
G06F 17/16

(21)Application number : 07-032099
(22)Date of filing : 21.02.1995

(71)Applicant : HITACHI LTD
(72)Inventor : TANAKA TERUO
YAMAMOTO YUSAKU
HOJO YOSHIAKI
TAMAOKI YOSHIKO
SAKAKIBARA TADAYUKI
ASAYA MACHIKO
YASUDA YOSHIKO

(54) PARALLEL EXECUTING METHOD FOR REPETITIVE PROCESSING

(57)Abstract:
PURPOSE: To execute a repetitive processing such as an LU decomposing processing by parallel computers in a shorten time.
CONSTITUTION: For example, a (K)th repetitive processing consists of a processing P(K) which can not be performed in parallel and a processing Q(K) which can be performed in parallel, the processing Q(K) uses the result of the processing P(K), and a (K+1)the processing P(K+1) uses the result of a processing QS(K) as part of the (K)th processing Q(K). In this case, the processing Q (K) is decomposed into processings Q1(K)-QP(K) as many as the number P of processors, and the processing QS(K) is included in the processing Q1(K). The processing Q1(K) and processing P(K+M) are executed in order by the same processor and other processing Q2 (K)-Qn(K) are executed by other (P-1) processors is parallel. The said processings are repeated thereafter each time of processes of all the processors end.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12) 公開特許公報（A）　(11)特許出願公開番号

特開平8－227405

(43)公開日　平成８年(1996)９月３日

(51)Int.Cl.[6]		識別記号	庁内整理番号	ＦＩ			技術表示箇所
Ｇ０６Ｆ	15/16	３９０		Ｇ０６Ｆ	15/16	３９０Ｚ	
	17/12				15/324		
	17/16				15/347	Ｋ	

審査請求　未請求　請求項の数14　ＯＬ　（全 16 頁）

(21)出願番号　特願平7－32099

(22)出願日　平成７年(1995)２月21日

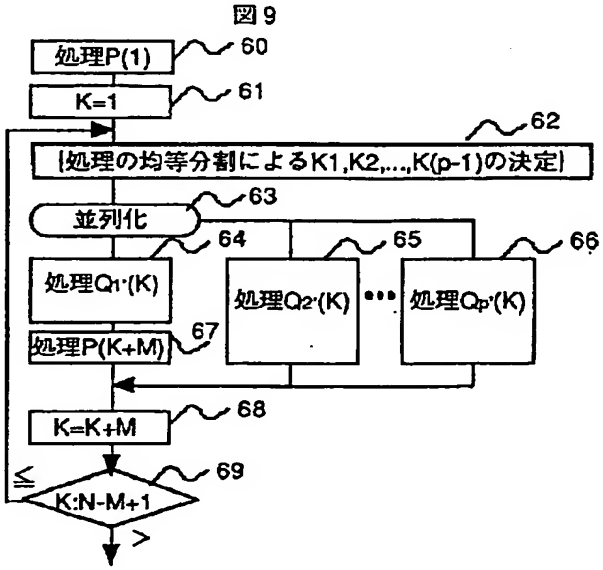
(71)出願人　000005108
株式会社日立製作所
東京都千代田区神田駿河台四丁目６番地
(72)発明者　田中　輝雄
東京都国分寺市東恋ケ窪１丁目280番地
株式会社日立製作所中央研究所内
(72)発明者　山本　有作
東京都国分寺市東恋ケ窪１丁目280番地
株式会社日立製作所中央研究所内
(72)発明者　北城　芳章
東京都江東区新砂一丁目６番27号　株式会社日立製作所公共情報事業部内
(74)代理人　弁理士　薄田　利幸

最終頁に続く

(54)【発明の名称】　反復処理の並列実行方法

(57)【要約】
【目的】ＬＵ分解処理などの反復処理を並列計算機でより短時間に実行可能にする。
【構成】Ｋ回目の反復処理が、並列化不可能な処理Ｐ（Ｋ）と並列化可能な処理Ｑ（Ｋ）とから構成され、処理Ｑ（Ｋ）は、処理Ｐ（Ｋ）の結果を利用し、かつ、Ｋ＋１回目の処理Ｐ（Ｋ＋１）はＫ回目の処理Ｑ1（Ｋ）中の一部の処理Ｑs（Ｋ）の結果を利用する時、処理Ｑ（Ｋ）を、プロセッサの台数 Pと同数の処理Ｑ1（Ｋ）からＱP（Ｋ）に分解し、かつ、処理Ｑ1（Ｋ）には、処理ＱS（Ｋ）を含ませる。処理Ｑ1（Ｋ）と処理Ｐ（Ｋ＋Ｍ）を同一のプロセッサで順次実行し、これと並行して、他の処理Ｑ2（Ｋ）からＱn（Ｋ）を他のP-1台のプロセッサで実行する。以後、全プロセッサの処理が終了する毎に以上の処理を繰り返す。

図９



【特許請求の範囲】
【請求項１】第１の処理と、その実行結果を使用する第２の処理とを反復して実行する反復処理であって、第２の処理は、並列に実行可能な複数の部分処理からなり、第１の処理は、該反復処理のある繰返し時における第２の処理の実行結果をその反復処理の次の繰返し時に使用するものを並列計算機で実行させる方法であって、
（ａ）該並列計算機を構成する複数のプロセッサのいずれか一つにより、該反復処理の第１の繰返しに対して第１の処理を実行させ、
（ｂ）一つの処理群が、第２の処理に含まれる上記複数の部分処理の内、第１の処理が次の繰返し時に使用するデータを生成する一部の複数の部分処理と該第１の処理とを少なくとも含むように、かつ、他の処理群が、第２の処理に含まれる上記複数の部分処理の内、該一部の複数の部分処理以外の他の複数の部分処理の一部を含むように、該第１の処理と第２の処理に含まれた上記複数該複数の部分処理とを、上記複数のプロセッサの数に等しい数の複数の処理群に分割し、
（ｃ）該第１の繰返しに対して、該ステップ（ａ）の実行結果を使用して該一部の複数の部分処理を実行し、該次の繰返しに対して、該一部の複数の部分処理のこの実行結果を使用して第１の処理を実行するように、該一部の複数の部分処理と該第１の処理を少なくとも含む上記一つの処理群を、該複数のプロセッサのいずれか一つにより実行し、
（ｄ）該他の複数のプロセッサの各々により、該複数の処理群の一つを、該第１の繰返しに対して該ステップ（ａ）の実行結果を使用して、かつ、該一つのプロセッサによるステップ（ｃ）での上記実行と並行して実行し、
（ｅ）各プロセッサによる、ステップ（ｃ）または（ｄ）の実行の終了後、上記ステップ（ｃ）から（ｅ）を繰返し、その繰返しにおいては、（ｅ１）該第２の処理を該次の繰返し以降の繰返しに対して実行され、かつ、該第１の処理を該次の繰返し以降の繰返しの次の繰返しに対して実行されるように、ステップ（ｃ）から（ｄ）を実行し、（ｅ２）上記繰返しをする前にステップ（ｃ）の実行の結果得られた、第１の処理の実行結果を、上記繰返しをした後に実行されるステップ（ｄ）において第２の処理により利用する反復処理の並列実行方法。
【請求項２】ステップ（ｅ）による繰返しのときに、ステップ（ｂ）も実行した上で繰り返す請求項１記載の反復処理の並列実行方法。
【請求項３】ステップ（ｂ）は、該一部の複数の部分処理に付加して該一つのプロセッサにより実行すべき他の一部の複数の部分処理が、該一つの処理群に属するように、該分割を実行するステップを有する請求項１記載の反復処理の並列実行方法。
【請求項４】ステップ（ｂ）は、上記一部の複数の部分処理と上記第１の処理の実行に必要な第１の処理時間と、該他の複数の部分処理の各々の実行に必要な第２の処理時間とに基づいて、該複数の処理群の実行時間がほぼ等しくなるように、上記分割を行なうステップを有する請求項１記載の反復処理の並列実行方法。
【請求項５】上記反復処理は、Ｎ次元連立一次方程式の係数行列をＬＵ分解する処理であり、上記第１の処理はピボット処理を含み、上記第２の処理は係数行列の消去更新処理を含む請求項１から４のいずれか一つに記載の反復処理の並列実行方法。
【請求項６】ステップ（ｅ）は、各プロセッサにおける処理の終了を検出した後、該一つのプロセッサによるステップ（ｃ）の繰返しおよび該他の複数のプロセッサの各々によるステップ（ｄ）の繰返しを同時に起動するステップを有する請求項１記載の反復処理の並列実行方法。
【請求項７】ステップ（ｅ）は、
各プロセッサにより、そのプロセッサでの処理の終了後に、他の複数のプロセッサの各々での処理の終了を判別し、
各プロセッサにより、他のプロセッサの各々での処理の終了が検出されたときに、そのプロセッサでの、ステップ（ｃ）または（ｄ）の繰返しを起動するステップを有する請求項１記載の反復処理の並列実行方法。
【請求項８】該並列計算機は該複数のプロセッサにより共有された主記憶をさらに有し、
該方法は、
（ｆ）各プロセッサが該ステップ（ｃ）または（ｄ）の実行を終了したとき、その終了を示す情報をそのプロセッサにより該主記憶に書き込むステップをさらに有し、
該ステップ（ｅ）は、各プロセッサから該主記憶に書き込まれた、そのプロセッサでの処理の終了を示す情報に基づいて行なわれる請求項７記載の反復処理の並列実行方法。
【請求項９】ステップ（ｆ）は、
ステップ（ｃ）の実行の終了後、該一つのプロセッサによりそのプロセッサでの処理の終了を示す情報を該主記憶に書き込み、
該他の複数のプロセッサの各々により、ステップ（ｄ）による、いずれかの部分処理群の実行の終了後に、そのプロセッサでの処理の終了を示す情報をその各プロセッサにより該主記憶に書き込むステップを有し、
ステップ（ｅ）は、
該一つのプロセッサにより、上記終了を示す情報を書き込んだ後に、該他の複数のプロセッサの各々による、終了を示す情報が該主記憶に書き込まれたか否かを該第一つのプロセッサにより監視し、
該他の複数のプロセッサの各々による、上記終了を示す情報の書き込みの後に、該他の複数のプロセッサの以外

のプロセッサの各々による、終了を示す情報が該主記憶に書き込まれたか否かを、該他の複数のプロセッサの各々により監視することにより、該複数のプロセッサの各々により、他のプロセッサでのステップ（ｃ）または（ｄ）での処理の実行が終了したか否かを判別するステップとを有し、
ステップ（ｅ）は、該ステップ（ｅ）による各プロセッサ別の、他プロセッサの終了の判別結果に基づいて、各プロセッサ毎に、ステップ（ｃ）から（ｄ）の繰返しの開始時期を決定するステップを有する請求項８記載の反復処理の並列実行方法。
【請求項１０】第１の処理と、その実行結果を使用する第２の処理とを反復して実行する反復処理であって、第２の処理は、並列に実行可能な複数の部分処理からなり、第１の処理は、該反復処理のある繰返し時における第２の処理の実行結果をその反復処理の次の繰返し時に使用するものを並列計算機で実行させる方法であって、
（ａ）該並列計算機を構成する複数のプロセッサのいずれか一つにより、該反復処理の第１の繰返しに対して第１の処理を実行させ、
（ｂ）一つの処理群が、第２の処理に含まれる上記複数の部分処理の内、第１の処理が次の繰返し時に使用するデータを生成する一部の複数の部分処理と該第１の処理とを少なくとも含むように、かつ、他の処理群が、第２の処理に含まれる上記複数の部分処理の内、該一部の複数の部分処理以外の他の複数の部分処理の一部を含むように、第２の処理に含まれた上記複数該複数の部分処理と第１の処理とを上記複数のプロセッサの数に等しい数の複数の処理群に分割し、
（ｃ）該第１の繰返しに対して、該ステップ（ａ）の実行結果を使用して該一部の複数の部分処理を実行し、該次の繰返しに対して、該一部の複数の部分処理のこの実行結果を使用して第１の処理を実行するように、該一部の複数の部分処理と該第１の処理を少なくとも含む上記一つの処理群を、該複数のプロセッサのいずれか一つにより実行し、
（ｄ）該他の複数のプロセッサの各々により、該複数の処理群の一つを、該第１の繰返しに対して、該ステップ（ａ）の実行結果を使用して、かつ、該一つのプロセッサによるステップ（ｃ）での上記実行と並行して実行し、
（ｅ）各プロセッサにおいて、そのプロセッサによるステップ（ｃ）または（ｄ）の実行が終了した時点で、そのプロセッサにより上記（ｃ）または（ｄ）を繰返し、その繰返しにおいては、（ｅ１）ステップ（ｃ）または（ｄ）で実行した処理群とは異なる他の一つの理群をそのプロセッサで実行するように、ステップ（ｃ）または（ｄ）を実行し、（ｅ２）当該他の処理群が第２の処理の部分処理群を含んでいるときには、それらの部分処理群を該次の繰返し以降の繰返しに対して実行し、当該他の処理群が第１の処理を含んでいるときには、その第１の処理を該次の繰返し以降の繰返しの次の繰返しに対して実行し、（ｅ３）そのプロセッサがステップ（ｃ）または（ｄ）で実行した処理群が第１の処理を含んでいないときには、他のプロセッサから通知される、第１の処理の実行結果の通知を確認した後に、上記繰返しを開始するステップを有する反復処理の並列実行方法。
【請求項１１】ステップ（ｅ）による該繰返しは、各プロセッサがステップ（ｅ）による繰返し毎に、順次異なる処理群を実行するように行なわれる請求項１０記載の反復処理の並列実行方法。
【請求項１２】いずれか一つのプロセッサによりステップ（ｃ）によりいずれかの繰返しに対して上記一つの処理群を実行したときに、その実行結果を他の複数のプロセッサに通知し、
他の複数のプロセッサの各々は、上記実行結果が、該一つのプロセッサから通知されたのを確認してから、その繰返しに対する、該複数の処理群の内のいずれかの他の一つの処理群を実行し、
該一つの処理群を上記繰返しの次の繰返しに対して実行する、さらに他のプロセッサに、該他の複数のプロセッサの各々により該他の処理群の実行結果を通知し、
該さらに他のプロセッサでは、そのプロセッサ以外の複数のプロセッサから上記繰返しに対する上記複数の処理群の実行結果の受信を、当該さらに他のプロセッサにより確認してから、上記さらに次の繰返しに対して、該一つの処理を開始するステップをさらに有する請求項１０記載の反復処理の並列実行方法。
【請求項１３】該並列計算機は、該複数のプロセッサを相互に接続するセットワークと、各プロセッサに対応して設けられ、そのプロセッサで実行されるプログラムとデータを保持するローカルメモリとを有する分散記憶型並列計算機であり、
上記一つの処理の実行結果の通知と上記他の処理群の実行結果の通知は、このネットワークを介して行なわれる請求項１２記載の反復処理の並列実行方法。
【請求項１４】第１の処理と、その実行結果を使用する第２の処理とを反復して実行する反復処理であって、第２の処理は、並列に実行可能な複数の部分処理からなり、第１の処理は、該反復処理のある繰返し時における第２の処理の実行結果をその反復処理の次の繰返し時に使用するものを、それぞれ複数のプロセッサとそれらのプロセッサで共有される、プログラムとデータを保持するためのローカルメモリとを有する複数のクラスタと、それらを結合するネットワークからなる分散記憶型の並列計算機で実行させる方法であって、
（ａ）該複数のクラスタ内のいずれか一つに含まれた複数のプロセッサのいずれか一つにより、該反復処理の第１の繰返しに対して第１の処理を実行させ、
（ｂ）一つの処理群が、第２の処理に含まれる上記複数

の部分処理の内、第１の処理が次の繰返し時に使用するデータを生成する一部の複数の部分処理と該第１の処理とを少なくとも含むように、かつ、他の処理群が、第２の処理に含まれる上記複数の部分処理の内、該一部の複数の部分処理以外の他の複数の部分処理の一部を含むように、該第１の処理と第２の処理に含まれた上記複数該複数の部分処理とを、上記複数のクラスタの数に等しい数の複数の処理群に分割し、

（ｃ）該複数のクラスタの各々により、該複数の処理群の一つを他のクラスタによる他の処理群の実行と並行して実行し、各クラスタによる、一つの処理群の実行に当たっては、（ｃ１）その一つの処理群に上記一部の複数の部分処理と該第１の処理とが含まれているときには、その一部の複数の部分処理と第１の処理とを少なくとも含む一つの部分処理群と、第２の処理に含まれる上記複数の部分処理の内、該一部の複数の部分処理以外の他の複数の部分処理の一部をそれぞれ含む複数の他の部分処理群とに、該一つの処理群の処理を分割し、該一つの処理群に上記一部の複数の部分処理と第１の処理が含まれていないときには、第２の処理に含まれる上記複数の部分処理の内、該一部の複数の部分処理以外の他の複数の部分処理の一部をそれぞれ含む複数の他の部分処理群に、該一該一つの処理群の処理を分割し、（ｃ２）そのクラスタ内の複数のプロセッサにより、ステップ（ｃ１）による分割で得られた複数の部分処理群の一つを、そのクラスタ内の他のプロセッサによる他の部分処理群の実行と並行して実行し、（ｃ３）そのクラスタで実行した上記一つの処理群に上記第１の処理が含まれている場合、その第１の処理が属する一つの部分処理群を実行した一つのプロセッサから、他のクラスタに第１の処理の実行結果を通知し、

（ｄ）各クラスタにおいて、そのクラスタ内の各プロセッサによるステップ（ｃ１）の実行が終了した時点で上記（ｃ）を繰返し、その繰返しにおいては、（ｄ１）該複数の処理群のうち、そのクラスタによりステップ（ｃ）で実行した処理群とは異なる他の処理群をそのクラスタで実行するように、ステップ（ｃ）を繰返し、

（ｄ２）ステップ（ｃ）でそのクラスタが実行した処理群の中に、該第１の処理が含まれていない場合には、他のクラスタから通知されるその第１の処理の実行結果の通知を確認した後、該他の処理群の実行を開始し、（ｄ３）その繰返しにおいて実行する該他の処理群が、第２の処理に属する部分処理群を含んでいる場合には、該次の繰返し以降の繰返しに対してその部分処理群を実行し、かつ、該他の処理群が、第１の処理を含んでいるときには、該次の繰返し以降の繰返しの次の繰返しに対して第１の処理を実行するように、該他の処理群を実行する反復処理の並列実行方法。

【発明の詳細な説明】

【０００１】

【産業上の利用分野】本発明は、Ｎ次元連立一次方程式の係数行列を下三角行列と上三角行列に分解するＬＵ分解処理のように、同じ処理を異なるデータに対して反復して実行する反復処理を、複数のプロセッサを持つ並列計算機で実行させる方法に関する。

【０００２】

【従来の技術】各種の工業用製品の開発に数値シミュレーションが実用に供されている。このような数値シミュレーションでよく使用される処理は、Ｎ次元連立方程式ＡＸ＝ＢをガウスのＡはＮ行Ｎ列の係数行列、ＸとＢはＮ列のベクトルである。このガウスの消去法により上記方程式を解く処理の主要な部分は、係数行列Ａを下三角行列Ｌと上三角行列Ｕの積である行列ＬＵに分解するＬＵ分解処理である。

【０００３】大規模行列のＬＵ分解処理の実用化は進んでおり、特にスーパコンピュータ向けには、多段同時展開法が一般的である。これらの手法は、文献：村田、小国、唐木著スーパコンピュータ、丸善、９１～１０３頁、１９８５や文献：ブロッキングＬＵ分解法のＶＰ２０００シリーズ向けチューニングについて、情報処理学会第４２回（平成３年前期）全国大会予稿集第１分冊７１～７２頁、特開平４－２５９０６４および特開平６－７５９８８などに開示されている。

【０００４】とくに最近は、並列計算機の進歩に伴い、並列計算機を数値シミュレーションに応用する研究が盛んである。

【０００５】近年実用化されている並列計算機は、一般的に次のような３種類の形態をとる。図１は、複数台のプロセッサ２ないし４が主記憶１を共有する主記憶共有型並列計算機である。これは、現在のベクトル型スーパコンピュータの主流であり、スーパコンピュータの場合、各プロセッサ２ないし４は多数のデータをベクトルデータとしてまとめ、そのベクトルデータに同一の処理をパイプライン演算器を使用して連続的に施すベクトルプロセッサとなる。

【０００６】図２は、各々が演算処理装置１０と専用の主記憶９を持つ独立したを計算機６ないし８である。ここで、計算機６ないし８をプロセッサと呼ぶ。これらのプロセッサ６ないし８を相互結合網５で接続する。現在は、各プロセッサはスーパスカラなどの高性能マイクロプロセッサとメモリで構成されることが多い。

【０００７】図３は、主記憶１５および複数台のプロセッサ１６ないし１７から構成される主記憶共有型並列計算機１２ないし１４（図１と同様）をさらに相互結合網１１で接続した構成を持つ。これは、たとえば、特開平６－９６０３２で開示されている。図３の構成における各々の主記憶共有型計算機１２ないし１４をクラスタと呼ぶ。

【０００８】図２の形式の並列計算機でＬＵ分解処理を実行させる方法の例は、文献：星野著ＰＡＸコンピュー

タ、オーム社、１９８５や特開平３－１０５６９４などに開示されている。

【０００９】ＬＵ分解処理の概要および図１に示した並列計算機でＬＵ分解処理を実行させる従来の方法を図４、図５を用いて説明する。

【００１０】図４は、行列Ａに対するＬＵ分解処理のＫ回目のループ処理を説明するための図である。すなわち、行列Ａの各要素ａ（ｉ，ｊ）（すなわち、第ｉ行第ｊ列番目の要素）を、異なる処理を受ける要素群に分けて、それらの要素群を領域の形で示している。図中、Ｎは行列サイズである。ここでは、第１行から第Ｋ－１行および第１列から第Ｋ－１列までのＬＵ分解処理は終了している。各反復回では、高速化のために、一度にＭ列をまとめてＬＵ分解処理を行なういわゆる多段同時消去方法が使用される。このためには、Ｍは複数であるが、原理的にはＭは１でもよい。

【００１１】以下の説明のため、行列Ａの第Ｋ列から第Ｎ列ないし第Ｋ行から第Ｎ行の領域を５つに分け、それぞれに次のような名称を付ける。

【００１２】（１）Ｌ１（Ｋ）２２：第Ｋ列から第Ｋ＋Ｍ－１列および第Ｋ行から第Ｋ＋Ｍ－１行で構成される左下三角領域

（２）Ｕ１（Ｋ）２１：第Ｋ列から第Ｋ＋Ｍ－１列および第Ｋ行から第Ｋ＋Ｍ－１行で構成される右上三角領域

（３）Ｌ２（Ｋ）２３：第Ｋ列から第Ｋ＋Ｍ－１列および第Ｋ＋Ｍ行から第Ｎ行で構成される矩形領域

（４）Ｕ２（Ｋ）２４：第Ｋ＋Ｍ列から第Ｎ列および第Ｋ行から第Ｋ＋Ｍ－１行で構成される矩形領域

（５）Ａ（Ｋ）２５　：第Ｋ＋Ｍ列から第Ｎ列および第Ｋ＋Ｍ行から第Ｎ行構成される矩形領域

図５に、図１の並列計算機で実行可能な、従来技術によるＬＵ分解処理プログラムを示す。この図において、ＬＵ分解プログラムは、２重のＤＯループで構成されている。

【００１３】外側のループ２０は、Ｍ列づつまとめて処理するために、行列のサイズＮをＭで割ったＭ／Ｎ回だけ繰り返す。

【００１４】内側は２つＤＯループで構成され、それらの処理は以下のとおりである。なお、外側ループ２０の繰返し回数はＫとする。

【００１５】まず、最初の内側ループ（ＤＯ　１）の処理では、第Ｋ列から第Ｋ＋Ｍ－１列番目まで、つまりＵ１（Ｋ）２１、Ｌ１（Ｋ）２２およびＬ２（Ｋ）２３の領域に対して、各列ごとに、部分軸選択（いわゆるピボット処理）およびそれに対する消去更新処理を繰り返して行なう。これらの処理の概要は以下のとおりである。最初の内側ループ（ＤＯ　１）のループ繰返し回数はＫＭとする。但し、ＫＭはＫ、Ｍとは関係のない値である。

【００１６】（１）ピボット処理：ＫＭ列の第ＫＭ行から第Ｎ行までの要素から最大値を検索し、検索した最大値を含む行と第ＫＭ行との行交換を行なう。すなわち、それらの行の第ＫＭ列から第Ｋ＋Ｍ－１列番目の要素を交換する。

【００１７】（２）消去更新処理：第ＫＭ＋１列から第Ｋ＋Ｍ－１列および第ＫＭ＋１行から第Ｎ行により構成される矩形領域内の各要素ａ（ｉ，ｊ）に対して、消去更新処理ａ（ｉ，ｊ）＝ａ（ｉ，ｊ）－ａ（ＫＭ，ｊ）×ａ（ｉ，ＫＭ）を行なう。

【００１８】このＤＯループの処理を、以下では、処理Ｐ（Ｋ）と呼ぶ。このＤＯループの処理では、最大値検索という逐次処理を繰り返して行なう必要があるため、このループ自体を並列化してもあまり性能が向上しない。

【００１９】次に、第２の内側ＤＯループ（ＤＯ　２）では、領域Ｕ２（Ｋ）および領域Ａ（Ｋ）に対して、次の処理を行なう。

【００２０】（１）領域Ｕ２（Ｋ）および領域Ａ（Ｋ）の行交換：前述のピボット処理において行交換処理を施したＭ組の行の第Ｋ＋Ｍ列から第Ｎ列の要素に対して、行交換処理を行なう。

【００２１】（２）領域Ｕ１（Ｋ）、領域Ｌ１（Ｋ）を用いた領域Ｕ２（Ｋ）の消去更新処理：この領域Ｕ２（Ｋ）内の各要素ａ（ｉ，ｊ）に対して、消去更新処理ａ（ｉ，ｊ）＝ａ（ｉ，ｊ）－ａ（ｋ，ｊ）×ａ（ｉ，ｋ）を行なう。ここで、ｋをＫからＫ＋Ｍ－１に順次変化させて、この計算を合計Ｍ回行なう。なお、ｉは、Ｋ＋１からＫ＋Ｍ－１の一つを１取り、ｊはＫ＋ＭからＮの一つの値を取る。したがって、この計算は、領域Ｕ１（Ｋ）、Ｌ１（Ｋ）内の要素を使用して、領域Ｕ２（Ｋ）内の要素を更新している。

【００２２】（３）領域Ｕ２（Ｋ）、Ｌ２（Ｋ）を用いた領域Ａ（Ｋ）内の要素の消去更新処理：消去更新処理ａ（ｉ，ｊ）＝ａ（ｉ，ｊ）－ａ（ｋ，ｊ）×ａ（ｉ，ｋ）を行なう。ここで、ｋをＫからＫ＋Ｍに順次変えて、この計算を合計Ｍ回行なう。ここで、ｉはｋ＋ＭからＮの一つであり、ｊはＫ＋ＭからＮの一つである。

【００２３】この第２の内側ループの処理を、以下では、処理Ｑ（Ｋ）と呼ぶ。

【００２４】これらの計算（１）から（３）は、領域Ｕ２（Ｋ）、Ａ（Ｋ）の各列ごとに独立に計算可能である。すなわち、各列毎に、上記処理（１）から（３）を実行する処理が基本の処理であり、処理Ｑ（ｋ）は、この単位処理を複数の列に対して集めたものとなっている。このため、このような各列毎の処理は、処理Ｑ（Ｋ）を構成する、互いに並列に実行可能な部分処理と言える。したがって、Ｑ（Ｋ）２８を列を単位に均等にｐ個にグループで分け、それぞれをＱ１（Ｋ）、Ｑ２（Ｋ）、、、ＱＰ（Ｋ）とし、ｐ台のプロセッサに分割して処理を行なうことにより、ｐ倍に近い台数効果を得

ることができる。

【００２５】このようなＬＵ分解処理のフローチャートは、図７の通りである。図７では、パラメタＫの初期値設定３５に続き、処理Ｐ（Ｋ）３６の実行後、並列化可能な処理Ｑ（Ｋ）の並列化３７を行ない、ｐ個の処理Ｑ１（Ｋ）３８ないし処理Ｑｐ（Ｋ）４０に分け、ｐ台のプロセッサでそれぞれの処理を分担し、並列に実行する。このとき、並列化された処理Ｑ（Ｋ）のあとに、処理を逐次にもどして、Ｋの更新処理４１と収束判定４２を行ない、次のＫ＋Ｍ回目の反復ループである処理Ｐ（Ｋ＋Ｍ）、さらに処理Ｑ（Ｋ＋Ｍ）を実行する。

【００２６】このように処理Ｑ（Ｋ）を複数の並列実行可能な処理に分けて実行するプログラムを得るには、最適化コンパイラの自動並列化機構を使用してソースプログラムの内の並列実行可能な部分を検出し、かつ、それらを自動的に並列実行可能なオブジェクトプログラムに変換する方法あるいはソースプログラムに並列指示行を追加し、これをコンパイラにより並列実行可能なオブジェクトプログラムに変換する方法が取られる。図５に示したプログラムは、ＤＯ　ＡＬＬタイプの並列化指示行２９を有する例である。このプログラムの場合、処理Ｑ（Ｋ）２８の並列化は並列指示行２９により各反復回ごとに毎回行なわれる。

【００２７】以上の説明から明かな通り、ＬＵ分解処理の基本的な処理の流れは、図６に示すように、パラメタＫの初期値設定３０およびそのパラメータの更新処理３３により制御し、処理Ｐ（Ｋ）３１と、処理Ｑ（Ｋ）３２を指定した条件３４（収束条件あるいは反復回数など）が成立するまで繰返し実行することである。図５のプログラムは、この処理を、図７に示す手順にしたがって図１の形式の並列計算機で実行可能にし、それでもってＬＵ分解処理の高速化を図っている。

【００２８】数値シミュレーションでは、ＬＵ分解処理に限らず、ほぼ同一の処理を、図６に示す手順にしたがって繰り返す反復処理が多く見受けられる。その場合は、Ｍは１でもよい。このような他の反復処理でも、処理Ｐが本来逐次的にしか実行できないとしても、図７に示す処理手順にしたがって処理Ｑを並列に実行することにより、このような他の反復処理も高速に実行できる。

【００２９】

【発明が解決しようとする課題】ＬＵ分解処理を図７に示した手順で処理した場合、処理Ｑ（Ｋ）が並列計算機を構成する複数のプロセッサにより並列に実行されるため、その処理の実行時間は、プロセッサ数が多いほど短縮される。しかし、各反復回ごとに処理Ｐ（Ｋ）は並列計算機を構成する一つのプロセッサにより実施されるため、この処理Ｐ（Ｋ）の実行時間が、プロセッサ数を多くしても、ＬＵ分解処理全体の処理時間を短縮するのを妨げている。

【００３０】いま仮に、対象とする行列のサイズＮを１０００としたとき、上記処理Ｑ（Ｋ）と処理Ｐ（Ｋ）は、それぞれＬＵ分解処理の総演算量の処理のおおよそ９８％および２％程度の演算量を占める。しかし、処理Ｐ（Ｋ）、処理Ｑ（Ｋ）の実行時の演算器の使用率が大幅に異なる。

【００３１】たとえば、図１の並列計算機が４台のプロセッサからなり、各プロセッサがベクトルプロセッサの場合、この処理Ｑ（Ｋ）は並列処理可能であり、この処理のために異なるベクトルプロセッサ内のパイプライン演算器も使用される上に、処理Ｑ（Ｋ）の計算処理は積和計算のため、各ベクトルプロセッサ内のパイプライン演算器の使用率が高い。一方、処理Ｐ（Ｋ）は、一台のベクトルプロセッサ内の演算器しか使用しない上に、この処理の主たる部分は最大値検索処理であり、必要とするデータ量に対して相対的に演算量が少なく、この一つのベクトルプロセッサ内の一部のパイプライン演算器しか使用せず、演算器の使用率は高くない。このため、上述の４台のベクトルプロセッサからなる並列計算機に対する推定例では、処理Ｑ（Ｋ）に対する処理Ｐ（Ｋ）の演算器利用率は約１／１０程度と低い。このため、処理Ｐ（Ｋ）と処理Ｑ（Ｋ）の処理時間の比は、

約１０：１２（＝２：９８／（１０×４））

となり、わずか２％の演算量の処理Ｐ（Ｋ）がＬＵ分解処理全体の処理時間の約半分を占める。したがって、４台のプロセッサを使用しているにもかかわらず最大２台分の性能向上しか得られない。

【００３２】また、図１のプロセッサ１等が、スーパースカラプロセッサからなり、かつプロセッサの総数が１００台程度である場合の推定例では、処理Ｑ（Ｋ）に対する処理Ｐ（Ｋ）の演算器利用率は約１／２である。このため、処理Ｐ（Ｋ）と処理Ｑ（Ｋ）の処理時間の比は、

約４：１（＝２：９８／（２×１００））

となり、わずか２％の演算量の処理Ｐ（Ｋ）が演算時間の約８０％を占めることがわかる。したがって、１００台のプロセッサを使用しているにもかかわらず１０台強の性能向上しか得られない。

【００３３】したがって、処理Ｐ（Ｋ）の実行時間が、ＬＵ分解処理の全体の処理時間に及ぼす影響をさらに減じることが望まれる。

【００３４】同じような問題は、ＬＵ分解処理に限らず、処理Ｐ（Ｋ）で例示される第１の処理と、その処理の実行後に実行すべき、処理Ｑ（Ｋ）で例示される並列実行可能な第２の処理とを繰返すとともに、この繰返しに当たっては、第２の処理の実行後に第１の処理を実行するように第１、第２のの処理を実行するときに生じる。

【００３５】言い替えると、従来の方法では、複数のプロセッサの内、一部のプロセッサは処理をしないで、他のプロセッサでの処理待ちの状態にある時間が長いこと

になる。

【００３６】したがって、本発明の目的は、本来は逐次処理されるべき第１の処理とその後に実行されるべき、並列実行可能な第２の処理とを有し、第２の処理の完了を待って再度第１、第２の処理を実行するように、第１、第２の処理を反復して実行すべき反復処理を、並列計算機で実行させる場合に、この反復処理全体の処理時間を軽減することが可能な反復処理の並列実行方法を提供することである。

【００３７】本発明の他の目的は、このような処理の実行時に、プロセッサが待ち状態にある時間を軽減することである。

【００３８】本発明のより具体的な目的は、ＬＵ分解処理を並列計算機で実行させる場合に、この処理全体の処理時間を軽減することが可能なＬＵ分解処理の並列実行方法を提供することである。

【００３９】本発明の他のより具体的な目的は、このＬＵ分解処理を実行しているときのプロセッサが待ち状態にある時間を減少させることである。

【００４０】

【課題を解決するためにの手段】上に説明したＬＵ分解処理では、外側ループの繰返し回数Ｋのときに、処理Ｐ（Ｋ）の結果を使用して実行される処理Ｑ（Ｋ）では、領域Ｕ２（Ｋ）と領域Ａ（Ｋ）の要素の更新後の値が計算される。これらの要素のうち、外側ループの次の繰返し時に実行される処理Ｐ（Ｋ＋Ｍ）の計算に使用する要素は、このＫ＋１回目における領域Ｕ１（Ｋ）、Ｌ１（Ｋ）、Ｌ２（Ｋ）に属すべき要素のみである。すなわち、図４において、第Ｋ＋Ｍ行から第Ｎ行と第Ｋ＋Ｍ列から第Ｋ＋２＊Ｍ－１列により構成される領域２６内の要素である。したがって、処理Ｐ（Ｋ＋Ｍ）は、この領域２６内の要素に関する処理Ｑ（Ｋ）（これをＱＳ（Ｋ）と呼ぶことにする）が終了後に実行を開始しなければならない。しかし、処理Ｐ（Ｋ＋Ｍ）は、領域Ａ（Ｋ）の内、この領域２６以外の領域内の要素に対する処理Ｑ（Ｋ）とは、並列に実行可能である。

【００４１】本発明では、このことを利用して、本来は逐次処理されるべき第１の処理（例えば、Ｐ（Ｋ）、以下括弧内は例示である）とその後に実行されるべき、並列実行可能な第２の処理（Ｑ（Ｋ））とを有し、第２の処理の完了を待って再度第１、第２の処理を実行するように、第１、第２の処理を反復して実行すべき反復処理を、並列計算機で実行させる場合に、ある繰返し（Ｋ回目）で実行される第２の処理（Ｑ（Ｋ））の内、次の繰返し（Ｋ＋Ｍ回目）で実行されるべき第１の処理（Ｐ（Ｋ＋Ｍ））が使用する処理結果を生成する部分処理（Ｑｓ（Ｋ））と、次の繰返し時の第１の処理（Ｐ（Ｋ＋Ｍ））とを同じプロセッサに割り当て、この第１の処理（Ｐ（Ｋ＋Ｍ））を、この部分処理（ＱＳ（Ｋ））に続いてそのプロセッサで実行させる。第２の処理（Ｑ（Ｋ））の内、この部分処理（ＱＳ（Ｋ））以外の部分を、残りのプロセッサの数Ｐ－１に等しい数の、並列に実行可能な処理（Ｑ２（Ｋ）からＱＰ（Ｋ））に分割し、それらの残りのプロセッサにより並列に実行させる。

【００４２】この際、上記部分処理（Ｑｓ（Ｋ））と第１の処理（Ｐ（Ｋ＋Ｍ））とを実行するプロセッサの処理時間が、他のプロセッサのそれより小さいときには、前者の処理時間が他のプロセッサのそれとほぼ同じになるように、第２の処理（Ｑ（Ｋ））の内、部分処理（ＱＳ（Ｋ））以外の一部もこの部分処理（ＱＳ（Ｋ））を割り当てたプロセッサに割り当てることがより望ましい。すなわち、第２の処理処理（Ｑ（Ｋ））を実施するのに必要な第１の処理（Ｐ（Ｋ））を、いずれかのプロセッサにより予め実行した後、上記のごとくにして各プロセッサに割り当てるべき第２の処理（Ｑ（Ｋ））の一部を決定し、決定された処理の割り当てに従って、各プロセッサで、部分処理（ＱＳ（Ｋ）もしくはＱｉ（Ｋ）（ｉ＝２からｐ－１））を実行させる。

【００４３】部分処理（ＱＳ（Ｋ））を割り当てられたプロセッサでは、この処理が終了すると、第１の処理（Ｐ（Ｋ＋Ｍ））を実行させる。この部分処理（ＱＳ（Ｋ）と第１の処理（Ｐ（Ｋ＋Ｍ））および他の部分処理（Ｑ２（Ｋ）からＱＰ（Ｋ））の実行の終了を待って、以上の処理を繰り返す。

【００４４】なお、本発明は、図２、図３の形式の並列計算機でも実施可能である。

【００４５】

【作用】次の反復回（（Ｋ＋Ｍ））の第１の処理（Ｐ（Ｋ＋Ｍ））は、第２の処理（Ｑ（Ｋ））の内、その第１の処理（Ｐ（Ｋ＋Ｍ））に先立って実行する必要がある部分処理（ＱＳ（Ｋ））以外の処理と並列に実行されるので、並列計算機内のプロセッサが待ち状態にある時間を軽減することが出来、計算機資源の有効な利用を図ることが出来る。それにより、この反復処理全体の処理時間を軽減することが可能になる。又、従来のごとく、第１の処理（Ｐ（Ｋ＋Ｍ））を第２の処理（Ｑ（Ｋ））と逐次に実行する従来の場合に比べて、この第１の処理（Ｐ（Ｋ＋Ｍ））の実行時間が、繰返し処理全体に及ぼす影響を減少できる。なお、第１の処理（Ｐ（Ｋ＋Ｍ））は、部分処理（ＱＳ（Ｋ））を割り当てられたプロセッサ内で、この処理の後に実行されるので、正常な結果を出力する。

【００４６】

【実施例】以下、本発明に係る反復処理の並列実行方法を図面に示したいくつかの実施例を参照してさらに詳細に説明する。なお、以下においては、同じ参照番号は同じものもしくは類似のものを表わすものとする。

【００４７】＜実施例１＞本実施例では、図１の主記憶共有型の並列計算機を使用した、ＬＵ分解処理の並列実行方法を示す。図８は本実施例で使用するプログラムの

例であり、図９はそのプログラムを図１の装置で実行する手順を示すフローチャートである。

【００４８】以下、本実施例を、図９のフローチャートに基づいて、従来例で示した図７と異なる部分について主に説明する。

【００４９】図５に関して説明した通り、ＬＵ分解処理での次反復回の処理Ｐ（Ｋ＋Ｍ）は、図４に示す行列Ａにおいて、点線で囲った領域２６に対して処理を実行する。したがって、前反復回の処理Ｑ（Ｋ）のうち、図４の行列Ａ内の領域２６である第Ｋ＋Ｍ列から第Ｋ＋２＊Ｍ－１列の更新処理ＱＳ（Ｋ）を処理Ｑ（Ｋ）から除いた残りの処理（Ｑ（Ｋ）－ＱＳ（Ｋ））と処理Ｐ（Ｋ＋Ｍ）は並列に実行可能である。処理Ｐ（Ｋ＋Ｍ）は処理ＱＳ（Ｋ）を終了すればいつでも実行可能となる。したがって、図９において、処理Ｑ（Ｋ）を並列化する処理６３を実行し、処理ＱＳ（Ｋ）を含む処理Ｑ１´（Ｋ）６４、処理Ｑ２´（Ｋ）６５ないし処理ＱＰ´（Ｋ）６６のｐ個に分割し、そのうち、処理Ｑ１´（Ｋ）６４を処理Ｐ（Ｋ＋Ｍ）６７と同じプロセッサで、その処理より前に実行することにより、処理Ｑ２´（Ｋ）６５ないし処理ＱＰ´（Ｋ）６６と処理Ｐ（Ｋ＋Ｍ）とを並列に実行することができるようになる。

【００５０】処理Ｐ（Ｋ＋Ｍ）６７をＫ回目の反復回で実行するために、６１、６８および６９で制御される反復ループに入る前に、Ｋ＝１のときの処理Ｐ（１）６０を１度だけ実行する必要がある。

【００５１】処理Ｑ１´（Ｋ）６４、処理Ｑ２´（Ｋ）６５ないし処理ＱＰ´（Ｋ）６６の分割は、処理Ｐ（Ｋ＋Ｍ）６７を含めて、全体の処理が均等になるように分割する必要がある。処理Ｑ（Ｋ）の分割は、各プロセッサに割り付ける列の数を変更することにより自由に扱うことができる。ここでは、処理Ｑ１´（Ｋ）６４で扱う列をＫ＋ＭからＫ１とし、処理Ｑ２´（Ｋ）６５で扱う列をＫ１＋１からＫ２とし、処理ＱＰ´（Ｋ）６６で扱う列をＫ（ｐ-1）からＮとする。

【００５２】この処理の均等分割によるＫ１、Ｋ２、、、Ｋ（ｐ-1）の決定６２の手順を説明する。

【００５３】まず、処理Ｐ（Ｋ＋Ｍ）に先だって実行する必要のある処理ＱＳ（Ｋ）は、最低、処理Ｑ１´（Ｋ）６４に含まれている必要がある。したがって、Ｋ１はＫ＋２＊Ｍ－１以上となる（条件１）。今仮に、処理Ｐ（Ｋ＋Ｍ）６７の演算量を処理Ｑ（Ｋ）における列の演算量に換算した時の列数をProwとする。処理Ｑ１´（Ｋ）６４と処理Ｐ（Ｋ＋Ｍ）６７の演算量の和を全体の１／ｐにし、かつ条件１を満たすためには、Ｋ１の値は次の式、

【００５４】

【数１】

$$K1=MAX(K+2M-1,\ K+M+(N-(K+M)+1-Prow)/p) \quad (1)$$

であらわすことができる。ここで、MAXは最大値を求める関数である。

【００５５】さらに、Ｋ２ないしＫ（ｐ-1）は処理Ｑ（Ｋ）から処理Ｑ１´（Ｋ）６４を除いた処理を、残りのｐ－１台のプロセッサで均等に分割すれば良い。したがって、Ｋ２ないしＫ（ｐ-1）は次の式、

【００５６】

【数２】

$$Ki=K1+(i-1)/(p-1)*(N-K1)+1 \quad (2)$$

であらわすことができる。ここで、ＫｉはＫ２ないしＫ（ｐ-1）をあらわす。

【００５７】処理の均等分割によるＫ１、Ｋ２、、、Ｋ（ｐ-1）の決定処理６２は、各反復回ごとに行なえば効率が良いが、対象とする行列は徐々に小さくなるので、数回の反復回ごとに値を更新するようにすることもできる。

【００５８】図８に示すプログラムは、プロセッサ台数Ｐが４の場合である。

【００５９】まず、Ｋ＝１のときの処理Ｐ（１）４５を実行し、ＤＯ９ループ４６で指示された反復処理を実行する。次に前述した式（式１）および（式２）により、処理を均等に分割するためにＫ１、Ｋ２、Ｋ３を決定４７する。

【００６０】プログラムの並列化記述は、ひとつのプロセッサのみに処理Ｐ（Ｋ＋Ｍ）５８を割り付けるので、ＳＥＣＴＩＯＮタイプの並列化指示行４８ないし５３を用いる。各プロセッサの処理は、各SECTON () 文で区切られた範囲を実行する。

【００６１】ＳＥＣＴＩＯＮ（1）４９では、処理Ｑ１´（Ｋ）５４と処理Ｐ（Ｋ＋Ｍ）５８をこの順番に実行する。ここで、処理Ｑ１´（Ｋ）５４は行列Ａの第Ｋ＋Ｍ列から第Ｋ１列までを担当する。ここで、Ｋ１は（式１）を用いて求める。（式１）のＫ１の定義により、処理Ｑ１´（Ｋ）５４は前述した処理ＱＳ（Ｋ）を含むことを保障することができる。

【００６２】ＳＥＣＴＩＯＮ（2）５０では、処理Ｑ２´（Ｋ）５５を実行する。処理Ｑ２´（Ｋ）５５は行列Ａの第（Ｋ１＋１）列から第Ｋ２列までを担当する。ここで、Ｋ２は（式２）を用いて求める。

【００６３】ＳＥＣＴＩＯＮ（3）５１では、処理Ｑ３´（Ｋ）５６を実行する。処理Ｑ３´（Ｋ）５６は行列Ａの第（Ｋ２＋１）列から第Ｋ３列までを担当する。ここで、Ｋ３も（式２）を用いて求める。

【００６４】ＳＥＣＴＩＯＮ（4）５２では、処理Ｑ４´（Ｋ）５７を実行する。処理Ｑ４´（Ｋ）５７は行列Ａの第（Ｋ３＋１）列から第Ｎ列までを担当する。ここで、Ｋ３は（式２）を用いて求める。

【００６５】図８に示したプログラムをコンパイルすることにより、図９に示した手順で実行されるオブジェク

トプログラムが生成される。このコンパイルにより、図８に示す各部分に対応するオブジェクト部分が生成されるとともに、これらのオブジェクト部分の実行を制御するために、並列化処理６３と同期処理６９を実行するコードがこのオブジェクトプログラムに追加される。このような目的のためのコンパイラはすでに知られている。

【００６６】この実施例により、処理Ｐを実質上、並列処理の中に組み込むことが可能となるために、処理Ｐの実行時間が実質的に（１－１／ｐ）に短縮される。この結果ＬＵ分解処理全体の処理時間が短縮される。例えば、図１の形式の並列計算機が４台のプロセッサを有し、かつ、各プロセッサがベクトルプロセッサの場合、行列のサイズＮを１０００で、処理Ｐと処理Ｑの実行時間がほぼ同程度の場合は、本実施例を用いることにより、ＬＵ分解処理の実行時間を最大３５％程度低減することができる。

【００６７】＜実施例２＞本実施例は、図１の主記憶共有型の並列計算機を使用した、、ＬＵ分解処理の並列実行方法の他の例を提供する。図１０は本実施例で使用するプログラムの例であり、図１１はこのプログラムを図１の装置で実行させるときの実行手順を示すフローチャートである。

【００６８】実施例１においては、図９で示したごとく、各反復ごとに毎回並列化処理６３を行なうことにより、処理Ｐ（Ｋ＋Ｍ）と次の反復回の処理Ｑ１´（Ｋ＋Ｍ）ないし処理ＱＰ´（Ｋ＋Ｍ）との順序関係を保証している。しかし、毎回行なう並列化処理６３はすべてのプロセッサ間の同期を取るために性能低下の要因となる可能性がある。本実施例では、きめ細かくＰｏｓｔ処理やＷａｉｔ処理を用いた同期処理を用いて順序保障処理を行ない、並列化処理を制御変数Ｋの反復ループの外側に移動させる方法を説明する。ここで用いるＰｏｓｔ処理やＷａｉｔ処理は、ＯＳの発行するＰｏｓｔ／Ｗａｉｔマクロではなく、コンパイラの並列制御ライブラリのひとつであるＰｏｓｔ／Ｗａｉｔライブラリであり、コストが小さい処理である。

【００６９】この処理手順を図１１を用いて図９と比較して説明する。まず、図９と同様に、処理Ｐ（１）１０１を実行する。次に、制御変数Ｋにより指示される反復ループの外側に並列化処理１０２を移動する。このため、処理の均等分割処理１０３ないし１０５、反復ループの制御変数Ｋの更新処理１１７ないし１１９および反復ループの判定処理１２０ないし１２２は、並列化された後、各プロセッサごとに実行させる。さらに、他のプロセッサの次の反復回の処理との順序関係を保障するために、プロセッサ間でＰｏｓｔ処理およびＷａｉｔ処理を実行する必要がある。

【００７０】Ｐｏｓｔ処理およびＷａｉｔ処理の設定すべき位置について説明する。基本的な方針はＰｏｓｔ処理をできるかぎり早く実行し、Ｗａｉｔ処理をできるかぎり遅く実行することである。このことにより、各プロセッサの実行の独立性を高め、動的な要因（たとえば、主記憶アクセス競合など）によるＷａｉｔ処理時の不要な同期待ちを軽減することができる。

【００７１】まず、処理Ｐ（Ｋ＋Ｍ）１０７を担当するプロセッサの実行順序を説明する。このプロセッサで実行する消去更新処理Ｑ１´（Ｋ）を処理ＱＳ（Ｋ）１０６と処理（Ｑ１´（Ｋ）－ＱＳ（Ｋ））１０８の２つに分解する。ここで、処理ＱＳ（Ｋ）１０６は、処理Ｐ（Ｋ＋Ｍ）１０７に先だって行なう行列Ａ内の第Ｋ＋Ｍ行から第Ｋ＋２＊Ｍ－１行の更新処理である。処理（Ｑ１´（Ｋ）－ＱＳ（Ｋ））１０８は、このプロセッサが担当する処理Ｑ（Ｋ）に関する残りの処理である。処理Ｐ（Ｋ＋Ｍ）１０７を実行することにより、他のプロセッサの次の反復ループの処理Ｑ２´（Ｋ＋Ｍ）１０９ないし処理ＱＰ´（Ｋ＋Ｍ）１１０を実行することができるので、ここで、Ｐｏｓｔ処理１１１およびＷａｉｔ処理１１５ないし１１６を実行する。ここで、Ｐｏｓｔ処理１１１は他のｐ－１台のプロセッサのＷａｉｔ処理と対応することとする。次に、処理（Ｑ１´（Ｋ）－Ｑ１Ｓ（Ｋ））１０８を実行する。次反復回の処理Ｐ（Ｋ＋２＊Ｍ）を実行するためには、その前に他のプロセッサの処理Ｑ２´（Ｋ）１０９ないし処理ＱＰ´（Ｋ）１１０を実行する必要があるので、ここで、Ｐｏｓｔ処理１１３ないし１１４およびＷａｉｔ処理１１２を実行する。ここで、Ｗａｉｔ処理１１２はｐ－１台のプロセッサからのＰｏｓｔ処理をすべて受けて成立することとする。

【００７２】このように、処理Ｑ１´（Ｋ）を処理ＱＳ（Ｋ）１０６と処理（Ｑ１´（Ｋ）－Ｑ１Ｓ（Ｋ））１０８の２つに分解することにより、Ｐｏｓｔ処理１１１は処理Ｐ（Ｋ＋Ｍ）１０７の直後に実行可能となり、処理（Ｑ１´（Ｋ）－Ｑ１Ｓ（Ｋ））１０８の時間だけＰｏｓｔ処理１１１を他のプロセッサのＷａｉｔ処理１１５ないし１１６より早く実行することが可能となる。

【００７３】図１０に示すプログラムでは、ＳＥＣＴＩＯＮタイプの並列化７０ないし７５は１回だけ行なう。各ＳＥＣＴＩＯＮ（）文とＳＥＣＴＩＯＮ（）文の間に各プロセッサで各々実行する制御変数Ｋの反復ループ７６ないし７９がある。それぞれの反復ループの中に各演算処理８０ないし８９がある。複数のＰｏｓｔ処理とＷａｉｔ処理を対応させるために、ここでは、イベント変数を引数とする。複数のイベント変数を持つＰｏｓｔ処理９０は、設定した数のイベントを発行し、Ｗａｉｔ処理９５ないし９７と対応する。複数のイベント変数を持つＷａｉｔ処理９４は、設定した数のイベントを対応するＰｏｓｔ処理９１ないし９３から受けることにより成立する。

【００７４】図１０に示したプログラムをコンパイルすることにより、図１１に示した手順で実行されるオブジェクトプログラムが生成される。このことは、実施例１

の場合と同様である。

【００７５】＜実施例３＞本実施例は、図２の主記憶分散型の並列計算機を使用した、ＬＵ分解処理の並列実行方法を提供する。図１３は本実施例で使用するプログラムの例であり、図１４はこのプログラムを図２の装置で実行させるときの実行手順を示すフローチャートである。

【００７６】このような並列計算機では、図２に示すように、相互結合網５を介して、プロセッサ６ないし８間でのデータの送受信処理が必要となる。このデータの送受信処理を用いて順序保証を行なう。以下では、メッセージ交換型の並列計算機を例に説明する。

【００７７】分散記憶型の並列計算機では、まず、データを分割配置する必要がある。ここでは、図１２に示すように、行列Ａを多段展開の幅Ｍごとに行を分割して、順次各プロセッサに分割する。このような分割では、各プロセッサが、順番にピボット処理Ｐを担当することになる。

【００７８】この処理手順を図１４を用いて説明する。図１１と比較して、図１４で異なっている部分は、（１）同期処理のＰｏｓｔ／Ｗａｉｔ処理の代わりに、データの送信／受信処理を伴うＳｅｎｄ処理およびＲｅｃｅｉｖｅ処理を用いること、（２）図１４の実施例では、つねに一つのプロセッサがピボット処理Ｐを担当していたのに対して、実施例３では、負荷をできるだけ均等にするために、各プロセッサが交互にピボット処理を担当することである。図１４では、２台のプロセッサ（ＰＥａとＰＥｂ）で処理を実行する場合を示す。もちろん、３台以上のプロセッサの場合も同様である。処理Ｑ（Ｋ）は、プロセッサ数である２つに均等に分割される。それぞれのプロセッサの担当する処理を処理Ｑａ（Ｋ）および処理Ｑｂ（Ｋ）とする。

【００７９】このとき、もし、データを送信するＳｅｎｄ処理とデータを受信するＲｅｃｅｉｖｅ処理が非同期に実行可能な場合は、Ｓｅｎｄ処理を可能なかぎり先行させて、Ｒｅｃｅｉｖｅ処理を同様に可能なかぎり遅らせて実行する必要がある。これは、図１１のＰｏｓｔ／Ｗａｉｔ処理を用いた時のスケジューリング方針と同様であり、処理Ｑａ（Ｋ）を処理Ｑｓ（Ｋ）１７０と処理（Ｑａ（Ｋ）－Ｑｓ（Ｋ））１７２とに分解することにより、実現可能である。

【００８０】まず、それぞれのプロセッサでは、１６０、１６１において自プロセッサがどちらのプロセッサであるかを判断する。これは、一般にたとえばＵＮＩＸコマンドのWHOAMIなどのような判別ライブラリを事前に用意しておく。ここでは、それぞれ自プロセッサであると判断した１６２および１６３とする。

【００８１】次に、プロセッサＰＥａで、処理Ｐ（１）１６６を行ない、生成した領域Ｌ２（図４）のデータをプロセッサＰＥｂに送信するためにＳｅｎｄ処理１６７を行なう。

【００８２】一方、プロセッサＰＥｂは、このデータを受信するＲｅｃｅｉｖｅ処理１６８を実行する。以下、各プロセッサはそれぞれ制御変数Ｋの反復ループに入る。

【００８３】まず、プロセッサＰＥｂは、処理ＱＳ（Ｋ）１７３を実行し、引き続き、処理Ｐ（Ｋ＋Ｍ）１７４を実行する。これにより、Ｋ＋Ｍ回目の消去処理に用いる領域Ｌ２の生成が可能となるために、ただちにプロセッサＰＥａにＳｒｅｎｄ処理１７８を実行する。その後、残りの処理（Ｑｂ（Ｋ）－Ｑｓ（Ｋ））１７５を実行し、次反復回であるＫ＋Ｍステップに手順を進める。

【００８４】一方、プロセッサＰＥａでは、まず処理Ｑａ（Ｋ）１６９で担当するＱの領域の更新処理を行ない、その後に、Ｋ＋Ｍステップに必要なＬ２のデータをＲｅｃｅｉｖｅ処理１７７で受け取る。このとき、プロセッサＰＥｂ側のＳｅｎｄ処理１７８は可能なかぎり早く実行されているため、Ｒｅｃｅｉｖｅ処理１７７での受信待ちはあまりなく、すぐにデータ受信が可能となる。

【００８５】次にＫ＋Ｍステップに進む。今度は、プロセッサＰＥａ側がピボット処理Ｐ（Ｋ＋２＊Ｍ）１７１を担当する。以下の処理は、プロセッサＰＥｂのＫステップの手順と同等である。他方のプロセッサＰＥｂでもＫ＋Ｍステップの処理は、プロセッサＰＥａのＫステップの処理と同様に行なう。

【００８６】まず、処理ＱＳ（Ｋ＋Ｍ）１７０を実行し、引き続き、処理Ｐ（Ｋ＋２＊Ｍ）１７１を実行する。これにより、Ｋ＋２＊Ｍ回目の消去処理に用いる領域Ｌ２の生成が可能となるために、ただちにプロセッサＰＥｂにＳｅｎｄ処理１７９を実行する。その後、残りの処理（Ｑａ（Ｋ＋Ｍ）－Ｑｓ（Ｋ＋Ｍ））１７２を実行し、次反復回であるＫ＋２＊Ｍステップに手順を進めるために、制御変数Ｋの更新処理１８１、判定処理１８３を行なう。

【００８７】一方、プロセッサＰＥｂでは、まず処理Ｑｂ（Ｋ）１７６で担当するＱの領域の更新処理を行ない、その後に、Ｋ＋２＊Ｍステップに必要なＬ２のデータをＲｅｃｅｉｖｅ処理１８０で受け取る。このとき、プロセッサＰＥａ側のＳｅｎｄ処理１７９は可能なかぎり早く実行されているため、Ｒｅｃｅｉｖｅ処理１８０での受信待ちはあまりなく、すぐにデータ受信が可能となる。その後、制御変数Ｋの更新処理１８２、判定処理１８４を行なう。

【００８８】図１３に示すプログラムでは、まず１３０では自プロセッサがどちらのプロセッサであるかを判断する。これは、一般にたとえばＵＮＩＸコマンドのWHOAMIなどのような判別ライブラリを事前に用意しておく。それぞれのプロセッサの処理は、条件文１３１ないし１３３で判別される。図１３においては、１３６以下１３

２の直前までがプロセッサＰＥａの処理で、１４８以下１３３の直前までがプロセッサＰＥｂの処理である。

【００８９】各プロセッサで交互にピボット処理を担当するために、まず、１回だけ、プロセッサＰＥａで、処理Ｐ（１）１３６を行ない、その結果をＳＥＮＤ処理１４５で送信し、プロセッサＰＥｂにおいてＲＥＣＥＩＶＥ処理１４８で受信する。その後、各プロセッサは、反復ループに入る。各反復ループＤＯ９１およびＤＯ９２は制御変数Ｋに関して、２回分の処理を行なっている。そのために、ＤＯ９１およびＤＯ９２ループの間隔の指定が、図１３の１３４、１３５に示すように、第１、２の実施例に比べて２倍となっている。

【００９０】Ｋ回目のステップでは、プロセッサＰＥｂにおいて、処理ＱＳ（Ｋ）１４１、処理Ｐ（Ｋ＋Ｍ）１４２および処理（Ｑｂ（Ｋ）－Ｑｓ（Ｋ））１４３を行ない、ＳＥＮＤ処理１４９でプロセッサＰＥａにピボット値を送信する。プロセッサＰＥａでは、平行して処理Ｑａ（Ｋ）１３７を行ない、ＲＥＣＥＩＶＥ処理１４６でピボット値を受信する。

【００９１】Ｋ＋Ｍ回目のステップでは、プロセッサＰＥａにおいて、処理ＱＳ（Ｋ＋Ｍ）１３８、処理Ｐ（Ｋ＋２＊Ｍ）１３９および処理（Ｑａ（Ｋ＋Ｍ）－Ｑｓ（Ｋ＋Ｍ））１４０の演算処理を行ない、ＳＥＮＤ処理１４７でプロセッサＰＥｂにピボット値を送信する。プロセッサＰＥｂでは、平行して処理Ｑｂ（Ｋ）１４４を行ない、ＲＥＣＥＩＶＥ処理１５０でピボット値を受信する。

【００９２】図１３に示したプログラムをコンパイルすることにより、図１４に示した手順で実行されるオブジェクトプログラムが生成される。このことは、実施例１の場合と同様である。

【００９３】＜実施例４＞本実施例は、図３に示した、複数のクラスタにより構成された主記憶分散型の並列計算機を使用した、ＬＵ分解処理の並列実行方法を提供する。図１５は本実施例で使用するプログラムの例であり、図１６はこのプログラムを図３の装置で実行させるときの実行手順を示すフローチャートである。

【００９４】このような並列計算機では、実施例３のように、データの送受信処理をもとに順序保証を行ない、さらに、各反復回ごとに、クラスタ内のプロセッサ間で第１、第２の実施例のような並列処理を実施する。

【００９５】実施例４の実施例３と異なる点は、各ステップ実行内で、クラスタ内のプロセッサ間の並列処理を実行することである。このときの手順は、ここでは図９を用いて説明した実施例１で示した方法を用いる。もちろん、実施例２で示した図１１の手順も利用することが可能である。

【００９６】本実施例では、２つのクラスタＣＬａとＣＬｂ、および各クラスタ内のプロセッサ数がｐ台と仮定する。以下、本実施例での処理手順を図１６を用いて説明する。

【００９７】図１４と比較して、図１６で異なる点は、（１）図１４におけるプロセッサＰＥａおよびＰＥｂがそれぞれ図１６のクラスタＣＬａおよびＣＬｂに対応すること、（２）各クラスタ内の各ステップの処理において、クラスタ内のｐ台のプロセッサにおいて、並列処理が行なわれることの２点である。図１６では、２台のクラスタ（ＣＬａとＣＬｂ）で処理を実行する場合を示す。もちろん、３台以上のクラスタの場合も同様である。

【００９８】このとき、もし、データを送信する送信Ｓｅｎｄ処理とデータを受信するＲｅｃｅｉｖｅ処理が非同期実行可能な場合は、Ｓｅｎｄ処理を可能なかぎり先行させて、Ｒｅｃｅｉｖｅ処理を同様に可能なかぎり遅らせて実行する必要がある。これは、図１４におけるＳｅｎｄ処理／Ｒｅｃｅｉｖｅ処理を用いた時のスケジューリング方針と同様である。

【００９９】最初に、クラスタＣＬａでピボット処理Ｐ（１）２５２を行ない、生成した領域Ｌ２（図４参照）のデータをクラスタＣＬｂに送信するためにＳｅｎｄ処理２５３を実行する。

【０１００】一方、クラスタＣＬｂはこのデータを受信するＲｅｃｅｉｖｅ処理２５４を実行する。以下、各クラスタは、それぞれ制御変数Ｋの反復ループに入る。制御変数Ｋの制御は、２７９ないし２８２で行なう。処理Ｑ（Ｋ）は、クラスタ数である２つに均等に分割される。それぞれのクラスタの担当する処理を処理Ｑａ（Ｋ）および処理Ｑｂ（Ｋ）とする。

【０１０１】まず、クラスタＣＬｂは、Ｋ回目の反復処理において、クラスタ内のｐ台のプロセッサを用いて並列処理２５６を行なう。このときの並列処理の分割の方法は、実施例１から実施例３にしてしたのと同一の方法で可能である。

【０１０２】ピボット処理Ｐ（Ｋ＋Ｍ）２６３を担当するプロセッサでは、処理ＱＳ（Ｋ）２６２を実行し、引き続き、ピボット処理Ｐ（Ｋ＋Ｍ）２６３を実行する。これにより、Ｋ＋Ｍ回目の消去処理に用いる領域Ｌ２の生成が可能となるために、ただちにクラスタＣＬａにＳｅｎｄ処理２６８を実行する。その後、残りの処理（Ｑｂ（Ｋ）－Ｑｓ（Ｋ））２６４を実行する。クラスタＣＬｂ内の他のプロセッサは、並列にそれぞれ処理Ｑｂ2（Ｋ）２６５および処理Ｑｂｐ（Ｋ）２６６を実行する。Ｋステップ内でクラスタ内でのプロセッサ間の同期は、並列化処理２５６で行なわれる。その後、Ｋ＋Ｍステップに手順を進める。

【０１０３】一方、クラスタＣＬａでは、まず処理Ｑａ1（Ｋ）２５９で担当するＱの領域の更新処理を行ない、その後に、Ｋ＋Ｍステップに必要なＬ２のデータをＲｅｃｅｉｖｅ処理２６７で受け取る。このとき、クラスタＣＬｂ側のＳｅｎｄ処理２６８は可能なかぎり早く

実行されているため、Ｒｅｃｅｉｖｅ処理２６７での受信待ちはあまりなく、すぐにデータ受信が可能となる。クラスタＣＬａ内の他のプロセッサは、並列にそれぞれ処理Ｑa2（Ｋ）２６０および処理Ｑap（Ｋ）２６１を実行する。Ｋステップ内でクラスタ内でのプロセッサ間の同期は、並列化処理２５５で行なわれる。その後、Ｋ＋Ｍステップに手順を進める。

【０１０４】次にＫ＋Ｍステップに進む。今度は、クラスタＣＬａ側がピボット処理Ｐ（Ｋ＋２＊Ｍ）２７０を担当する。以下の処理は、クラスタＣＬｂのＫステップの手順と同等である。

【０１０５】処理ＱＳ（Ｋ＋Ｍ）２６９を実行し、引き続き、ピボット処理Ｐ（Ｋ＋２＊Ｍ）２７０を実行する。これにより、Ｋ＋２＊Ｍ回目の消去処理に用いる領域Ｌ２の生成が可能となるために、ただちにクラスタＣＬｂにＳｅｎｄ処理２７７を実行する。その後、残りの処理（Ｑa1（Ｋ＋Ｍ）－Ｑs（Ｋ＋Ｍ））２７１を実行する。クラスタＣＬａ内の他のプロセッサは、並列にそれぞれ処理Ｑa2（Ｋ）２７２および処理Ｑap（Ｋ）２７３を実行する。Ｋ＋Ｍステップ内でクラスタ内でのプロセッサ間の同期は、並列化処理２５７で行なわれる。その後、Ｋ＋２＊Ｍステップに手順を進める。

【０１０６】一方、クラスタＣＬｂでは、まず処理Ｑb1（Ｋ＋Ｍ）２７４において、担当する領域の更新処理を行ない、その後に、Ｋ＋２＊Ｍステップに必要なＬ２のデータをＲｅｃｅｉｖｅ処理２７８で受け取る。このとき、クラスタＣＬａ側のＳｅｎｄ処理２７７は可能なかぎり早く実行されているため、Ｒｅｃｅｉｖｅ処理２７８での受信待ちはあまりなく、すぐにデータ受信が可能となる。クラスタＣＬｂ内の他のプロセッサは、並列にそれぞれ処理Ｑb2（Ｋ）２７５および処理Ｑbp（Ｋ）２７６を実行する。Ｋ＋Ｍステップ内でクラスタ内でのプロセッサ間の同期は、並列化処理２５８で行なわれる。その後、Ｋ＋２＊Ｍステップに手順を進める。

【０１０７】図１５に示すプログラムでは、２台のクラスタ（ＣＬａとＣＬｂ）でかつ各クラスタ内のプロセッサ数がそれぞれ２台の場合を示す。もちろん、３台以上のクラスタの場合、あるいは、クラスタ内のプロセッサ台数を３台以上にすることに可能である。図１５中、２０６以下２０２の直前までがクラスタＣＬａの処理で、２０８以下２０３の直前までがクラスタＣＬｂの処理である。ここでは、まず２００では自クラスタがどちらのクラスタであるかを判断する。これは、一般にたとえばＵＮＩＸコマンドのWHOAMIなどのような判別ライブラリを事前に用意しておく。

【０１０８】各クラスタで交互にピボット処理を行なうために、ＤＯ９１ループ２０４およびＤＯ９２ループ２０５は制御変数Ｋに関して、２回分の処理を行なっている。そのために、実施例３と同様に、ＤＯ９１およびＤＯ９２ループの間隔の指定が、実施例１、２に比べて２倍となっている。

【０１０９】各クラスタ内では、各反復ステップＫおよびＫ＋Ｍごとに、並列処理指示文２０９ないし２２４により、並列化の指示を行なう。

【０１１０】図１５の例では、クラスタＣＬａにおいては、ステップＫに対しては、２０９から２１２の間の並列指示行により並列処理が規定されており、演算処理２２５および２２６、ＲＥＣＥＩＶＥ処理２３７を行なう。ステップＫ＋Ｍでは、２１３から２１６の間の並列指示行により並列処理が規定されており、演算処理２２７ないし２２９および２３０、ＳＥＮＤ処理２３９を行なう。

【０１１１】クラスタＣＬｂにおいては、ステップＫに対しても、同様に２１７から２２０の間の並列指示行により並列処理が規定されており、演算処理２３１ないし２３４、ＳＥＮＤ処理２３８を行なう。ステップＫ＋Ｍでは、２２１から２２４の間の並列指示行により並列処理が規定されており、演算処理２３５および２３６、ＲＥＣＥＩＶＥ処理２４０を行なう。

【０１１２】なお、第３および実施例４では、この処理では、データの送受信のためのＳｅｎｄ処理やＲｅｃｅｉｖｅ処理を他の演算処理と並列に実行することが必要となる。もちろん、このＳｅｎｄ処理やＲｅｃｅｉｖｅ処理が専用の通信プロセッサで動作可能である場合は、それらの通信に要する処理時間を除いたのちに、演算を担当するプロセッサ内で演算量の処理を均等にする必要がある。

【０１１３】図１５に示したプログラムをコンパイルすることにより、図１６に示した処理手順が実行されるオブジェクトプログラムが生成されることは、実施例１の場合と同様である。

【０１１４】

【発明の効果】本来は逐次処理されるべき第１の処理とその後に実行されるべき、並列実行可能な第２の処理とを有し、第２の処理の完了を待って再度第１、第２の処理を実行するように、第１、第２の処理を反復して実行すべき反復処理を、並列計算機で実行させる場合に、プロセッサが待ち状態にある時間を軽減することが出来、計算機資源の有効な利用を図ることが出来る。それにより、この反復処理全体の処理時間を軽減することが可能になる。

【図面の簡単な説明】

【図１】従来の主記憶共有型並列計算機の概略構造を示す図。

【図２】従来の主記憶分散型並列計算機の概略構造を示す図。

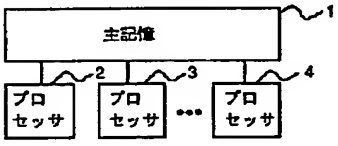
【図３】従来のクラスタ結合型並列計算機の概略構造を示す図。

【図４】従来の多段同時消去ＬＵ分解法の概要を示す図。

【図５】従来の多段同時消去ＬＵ分解法のプログラム例を示す図。
【図６】従来の反復計算処理のフローチャート。
【図７】並列化された反復計算処理のフローチャート。
【図８】本発明の実施例１で使用するプログラム例を示す図。
【図９】実施例１による反復計算処理のフローチャート。
【図１０】実施例２で使用するプログラム例を示す図。
【図１１】本発明の実施例２による反復計算処理のフローチャート。
【図１２】分散記憶型計算機向けのデータの分割を示す図。
【図１３】本発明の実施例３で使用するプログラム例を示す図。
【図１４】実施例３による反復計算処理のフローチャート。
【図１５】本発明の実施例４で使用するプログラム例を示す図。
【図１６】実施例４による反復計算処理のフローチャート。
【符号の説明】
２０、３４、４２…ループ制御、２７、３１、３６…逐次的処理Ｐ（Ｋ）、２８、３２…並列化可能処理Ｑ（Ｋ）、２９…並列指示、３８ないし４０…並列化可能処理Ｑ（Ｋ）を分割した処理

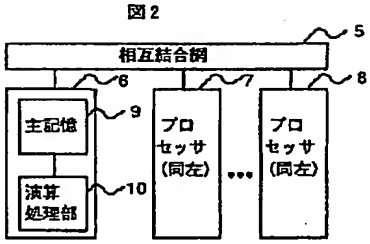
【図１】

図1



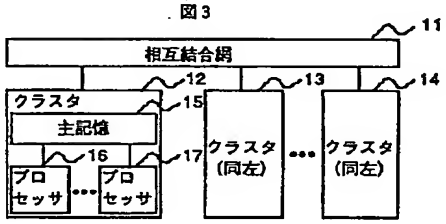
【図２】

図2



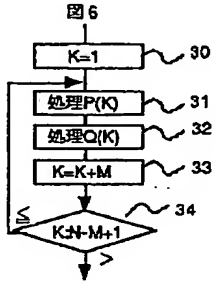
【図３】

図3



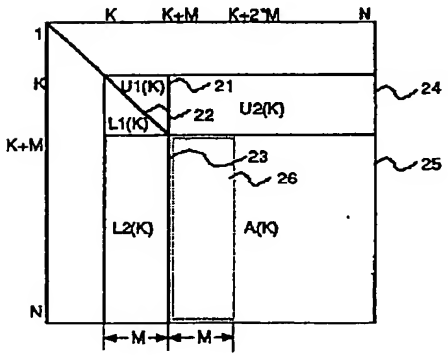
【図６】

図6



【図４】

図4



【図５】

図5

```
DO 9 K=1,N-M+1,M                                    20
  DO 1 KM=K,K+M-1                                   27
    ピボット処理(KM列中の最大値検索と行交換)          処理P(K)
    L1(K),U1(K),L2(K)領域の消去更新
1   CONTINUE
*POPTION PARALLEL                                   29
  DO 2 J=K+M, N                                     28
    U2(K)およびA(K)領域に対する行交換                 処理Q(K)
    U1(K),L1(K)領域を用いたU2(K)領域の消去更新
    U2(K),L2(K)領域を用いたA(K)領域の消去更新
2   CONTINUE
9 CONTINUE
```

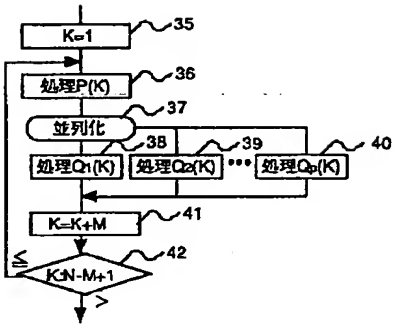
【図８】

図8

```
処理P(1)                                            45
  DO 9 K=1, N-M+1,M                                  46
[処理の均等分割によるK1,K2,K3の決定]                  47
*POPTION PARALLEL_SECTIONS                           48
*POPTION SECTION (1)                                 49
    処理Q1'(K)【J = K+M, K1 を担当】                  54
    処理P(K+M)                                       58
*POPTION SECTION (2)                                 50
    処理Q2'(K)【J = K1+1, K12を担当】                 55
*POPTION SECTION (3)                                 51
    処理Q3'(K)【J = K2+1, K3 を担当】                 56
*POPTION SECTION (4)                                 52
    処理Q4'(K)【J = K3+1, Nを担当】                   57
*POPTION END PARALLEL_SECTIONS                       53
 9 CONTINUE
```

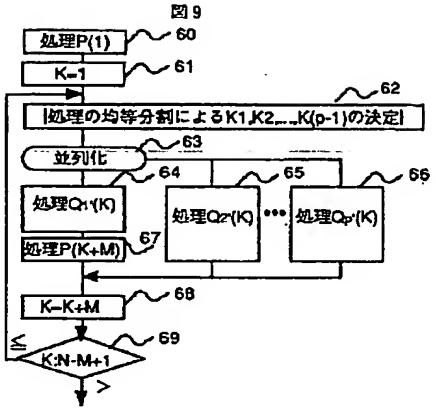
【図７】

図７



【図９】

図９



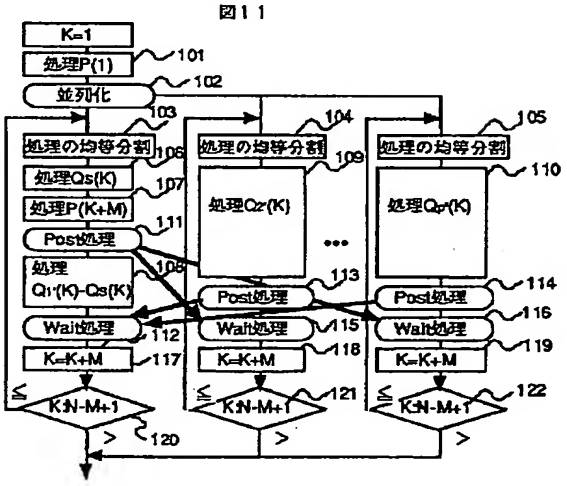
【図１０】

図１０

```
    処理 P(1)
*POPTION PARALLEL_SECTIONS                          70
*POPTION SECTION (1)                                71
    DO 9 1 K=1, N-M+1,M                             76
  [処理の均等分割によるK1の決定]                    80
      処理 Qs (K) [J = K+M, K+2*M-1 を担当]         84
      処理 P(K+M)                                   85
      CALL POST( EV1, EV2, EV3 )                    90
      処理 (Q1' (K)-Qs (K)) [J =K+2*M, K1 を担当]   86
      CALL WAIT( EV4, EV5, EV6 )                    94
9 1 CONTINUE
*POPTION SECTION (2)                                72
    DO 9 2 K=1, N-M+1,M                             77
  [処理の均等分割によるK1,K2の決定]                 81
      処理 Q2' (K) [J =K1, K2 を担当]               87
      CALL POST( EV4 )                              91
      CALL WAIT( EV1 )                              95
9 2 CONTINUE
*POPTION SECTION (3)                                73
    DO 9 3 K=1, N-M+1,M                             78
  [処理の均等分割によるK2,K3の決定]                 82
      処理 Q2' (K) [J =K2, K3 を担当]               88
      CALL POST( EV5 )                              92
      CALL WAIT( EV2 )                              98
9 3 CONTINUE
*POPTION SECTION (4)                                74
    DO 9 4 K=1, N-M+1,M                             79
  [処理の均等分割によるK3,Nの決定]                  83
      処理 Q2' (K) [J =K3, N を担当]                89
      CALL POST( EV6 )                              93
      CALL WAIT( EV3 )                              97
9 4 CONTINUE
*POPTION END PARALLEL_SECTIONS                      75
```

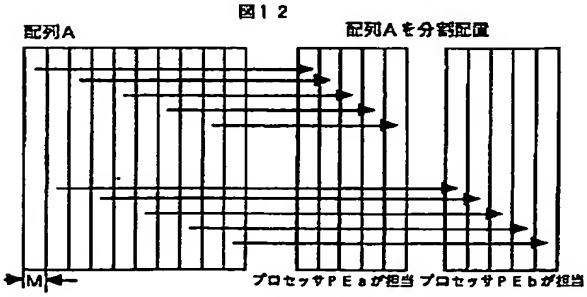
【図１１】

図１１



【図１２】

図１２



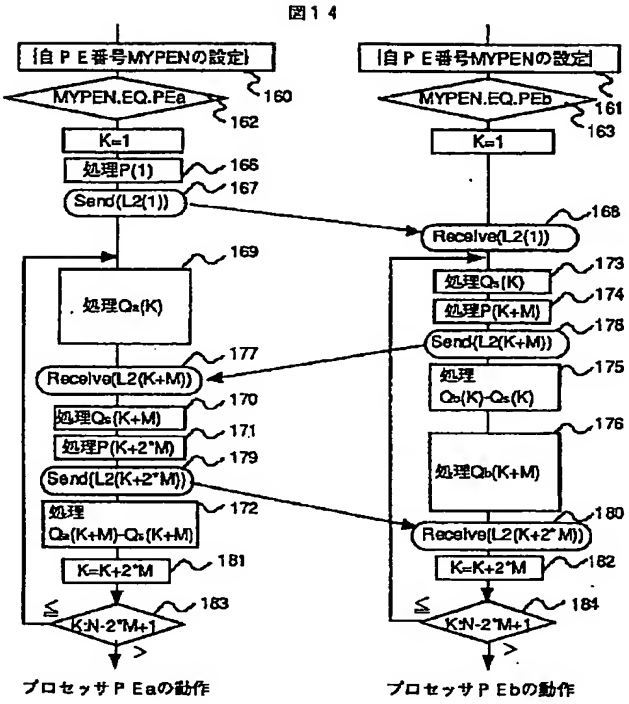
【図１３】

図１３

```
    [自ＰＥ番号MYPENの設定]                    130
    IF ( MYPEN.EQ.PEa ) THEN                   131
      処理 P(1)                                136
      CALL SEND( L2(1) )                       145
      DO 9 1 K=1, N-2*M+1, 2*M                 134
C Step K
        処理 Qa (K)                            137
        CALL RECEIVE( L2(K+M) )                148
C Step K+M
        処理 Qs (K+M)                          138
        処理 P(K+2*M)                          139
        CALL SEND( L2(K+2*M) )                 147
        処理 (Qa (K)-Qs (K))                   140
9 1   CONTINUE
    ELSE IF ( MYPEN.EQ.PEb ) THEN              132
      CALL RECEIVE( L2(1) )                    148
      DO 9 2 K=1, N-2*M+1, 2*M                 135
C Step K
        処理 Qs (K)                            141
        処理 P(K+M)                            142
        CALL SEND( L2(K+M) )                   149
        処理 (Qb (K)-Qs (K))                   143
C Step K+M
        処理 Qb (K+M)                          144
        CALL RECEIVE( L2(K+2*M) )              150
9 2   CONTINUE
    END IF                                     133
```

【図１４】

図１４



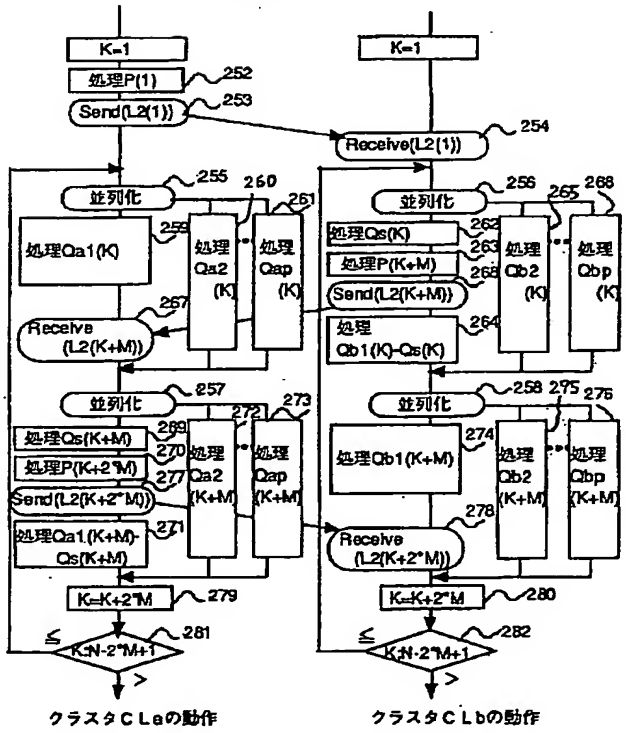
【図１５】

図１５

```
    [自クラスタ番号MYCLNの設定]                200
    IF ( MYCLN.EQ.CLa ) THEN
      処理 P(1)                                206
      CALL SEND( L2(1) )
      DO 9 1 K=1, N-2*M+1, 2*M                 204
C Step K
*POPTION PARALLEL_SECTIONS                     209
*POPTION SECTION (1)                           210
        処理 Qa1 (K)                           225
        CALL RECEIVE( L2(K+M) )                237
*POPTION SECTION (2)                           211
        処理 Qa2 (K)                           226
*POPTION END PARALLEL_SECTIONS                 212
C Step K+M
*POPTION PARALLEL_SECTIONS                     213
*POPTION SECTION (1)                           214
        処理 Qs (K+M)                          227
        処理 P(K+2*M)                          228
        CALL SEND( L2(K+2*M) )                 239
        処理 (Qa1 (K+M)-Qs (K+M))              229
*POPTION SECTION (2)                           215
        処理 Qa2 (K+M)                         230
*POPTION END PARALLEL_SECTIONS                 216
9 1   CONTINUE
    ELSE IF ( MYPEN.EQ.CLb ) THEN              202
      CALL RECEIVE( L2(1) )                    208
      DO 9 2 K=1, N-2*M+1, 2*M                 205
C Step K
*POPTION PARALLEL_SECTIONS                     217
*POPTION SECTION (1)                           218
        処理 Qs (K)                            231
        処理 P(K+M)                            232
        CALL SEND( L2(K+M) )                   238
        処理 (Qb1 (K)-Qs (K))                  233
*POPTION SECTION (2)                           219
        処理 Qb2 (K)                           234
*POPTION END PARALLEL_SECTIONS                 220
C Step K+M
*POPTION PARALLEL_SECTIONS                     221
*POPTION SECTION (1)                           222
        処理 Qb1 (K+M)                         235
        CALL RECEIVE( L2(K+2*M) )              240
*POPTION SECTION (2)                           223
        処理 Qb2 (K+M)                         236
*POPTION END PARALLEL_SECTIONS                 224
9 2   CONTINUE
    END IF                                     203
```

【図１６】

図１６



フロントページの続き

(72)発明者　玉置　由子
東京都国分寺市東恋ケ窪１丁目280番地
株式会社日立製作所中央研究所内
(72)発明者　榊原　忠幸
東京都国分寺市東恋ケ窪１丁目280番地
株式会社日立製作所中央研究所内
(72)発明者　朝家　真知子
東京都国分寺市東恋ケ窪１丁目280番地
株式会社日立製作所中央研究所内
(72)発明者　保田　淑子
東京都国分寺市東恋ケ窪１丁目280番地
株式会社日立製作所中央研究所内